Bike Safety Cam

Abstract:

In this project, I attempted to create a portable camera that could attach to a bicycle. The camera would take pictures whenever it received input that showed that the cyclist might be in danger, such as vibration input, acceleration input, and gyroscopic input. The camera would then save the images to be transferred to the user's phone along with the video. Ultimately, I couldn't completely achieve this goal, because I couldn't send the images or video to a user's phone, but I did get the camera to respond to sensor input.

Introduction:

I chose to build a bicycle safety camera because as an avid cyclist, I've had too many close calls where I was unable to hold anyone accountable. In December 2018, I was hit by a truck in broad daylight while I was in the bicycle lane. In that case, I was lucky because the truck driver stopped, and someone immediately called 911. But there are too many near misses to count where someone doesn't stop, so I wanted to make a tool that would hold drivers accountable and by doing so, hopefully make cyclists safer.

This was an interesting problem for me because I had never done anything with computer vision or movement data before. Interpreting the movement data required a level of mathematical knowledge that I hadn't anticipated, and I wasn't sure whether I could do it.

Project Description:

I build an acrylic box with a movable front and back side. The front and back side closed the box using magnets and the front side of the box had a cut out for the camera lens. Enclosed in the box was the project build including:

- An openMV M7 camera
- A DFRobot IO Expansion Shield for the openMV Cam M7
- An MPU-6050 (Gyroscope + Accelerometer + Temperature) Sensor Module
- A full breadboard
- An Xbee
- An Xbee Explorer
- A Raspberry Pi 3 Model B
- An Adafruit Pi Cobbler Plus with a breakout cable
- A DFRobot Digital Piezo Disk Vibration Sensor
- A 6mm tactile button switch
- A 10KΩ
- Many wires

Challenges:

The one of the hardest parts of the build was constructing an on button for the camera. Initially, it seemed like it should have been the easiest part of the project because we had done so may builds that involved button switches. What I didn't realize at the time was that we had never accounted for bouncing in our builds. When I tried to make an on/off button, the program would either shut down the camera too early, or it would just crash in the IDE. Once I accounted for the bounce, it worked fine.

Another challenge was determining what elements of the accelerometer and gyroscope data were relevant to the project. Initially, I thought that the gyroscope would handle position while the accelerometer would handle speed. Upon further reading, it seemed to be the opposite, so I edited the program to account for that and solely focused on the accelerometer reading from the Z axis and all the gyroscopic readings. I'm still not sure whether that was the right thing to do.

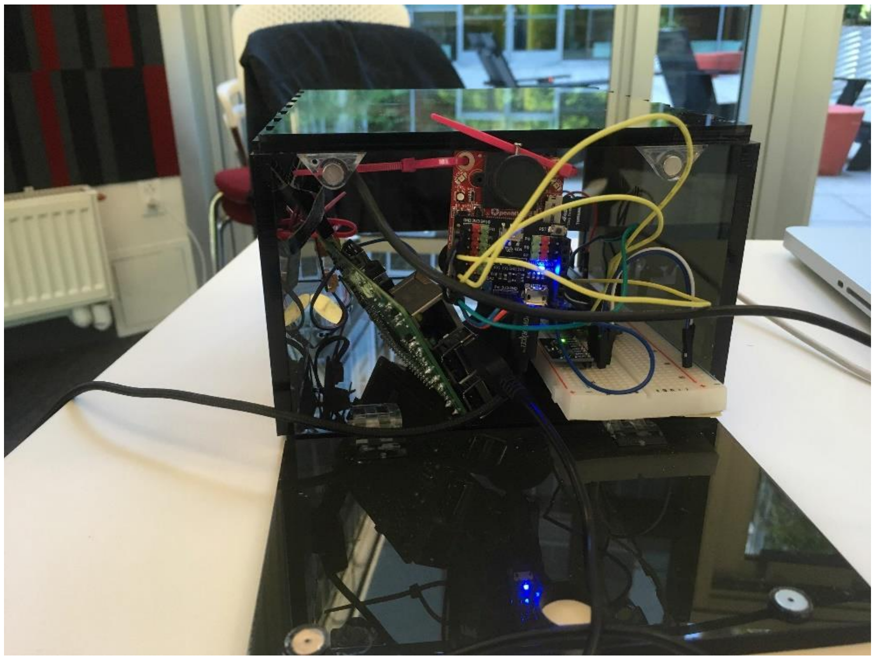
On the external side, one of my biggest challenges was dealing with edits to the box design. Every time I came up with a design, I had to make edits to account for issues that I hadn't known of initially. For example, my initial idea was to screw the Raspberry Pi and sensors in with M3 screws, but I later learned that acrylic could crack and there was no guarantee it wouldn't shatter. So, I had to edit the design to attach the devices in a different way. And I didn't realize that attaching all the sides of the box before installing the equipment would make it almost impossible to fix anything into place. Designing and planning out the box was one of my biggest challenges.

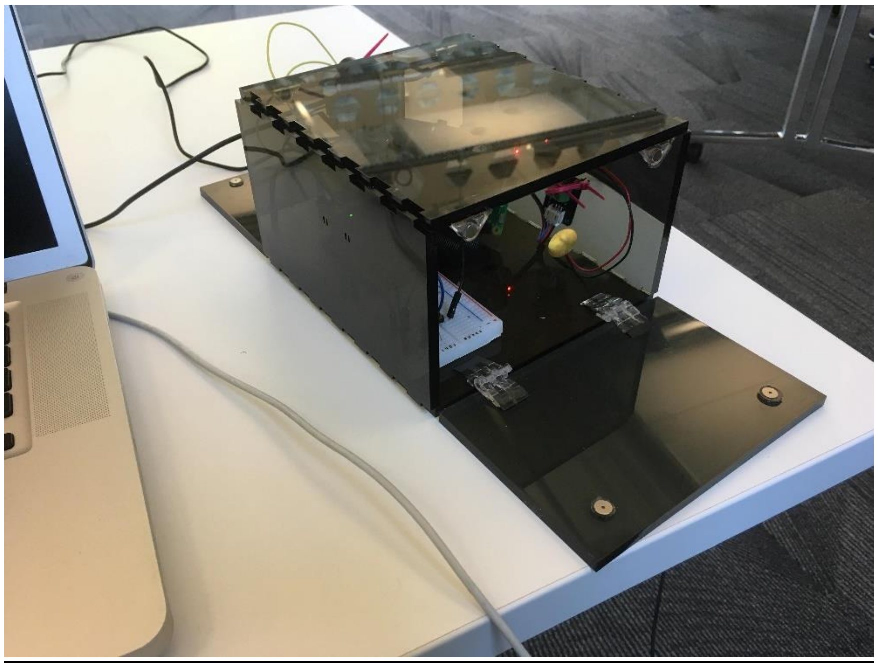
Design Decisions:

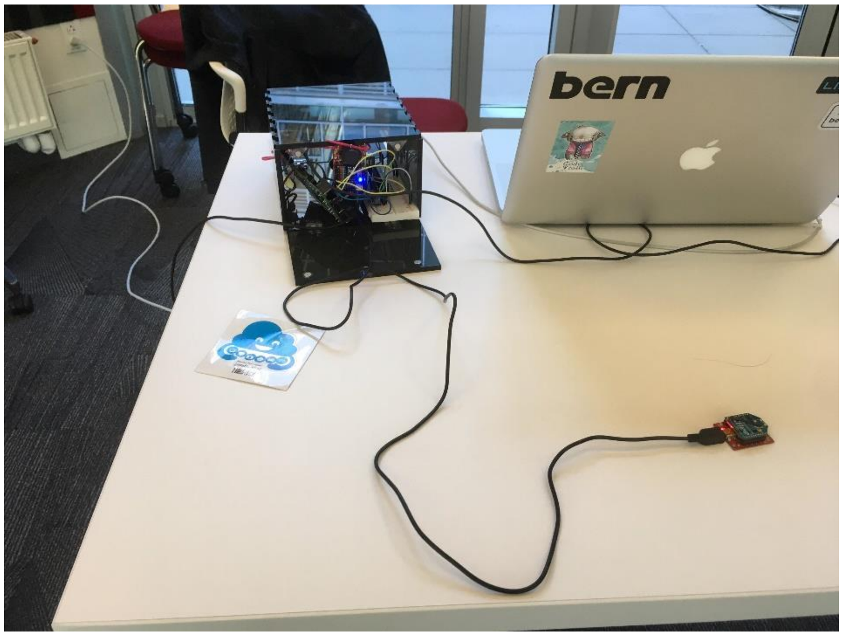
I had to first decide what material I wanted to make my container out of. I decided to use acrylic over wood because in a real-world situation, I would want the camera enclosed in plastic because rain happens, and wood would soak up the water. I decided to have the Raspberry Pi and breadboard at the bottom of the container so they would be side to side and keep each other from moving too much. Initially, I wanted the XBee on the bottom as well, but the USB cables attached to the Raspberry Pi, the openMV M7, and the XBee made it impossible.

Regarding the camera placement, I decided to have it on one of the shorter sides because, assuming you hung this from the handlebars, I didn't want it to be so long that it would interfere with the wheels. I decided to use hinges and magnets to close the box because I needed an easy way to remove all the components from the container once class was over. There are slots on the bottom, and sides of the case so I can attach the Raspberry Pi and the peripherals to the box. I put a hole in the front side of the box, so I can put the camera lens through and get an unobstructed view of everything and I attached it to the top of the box, so it wouldn't complete block the breadboard and Raspberry Pi.

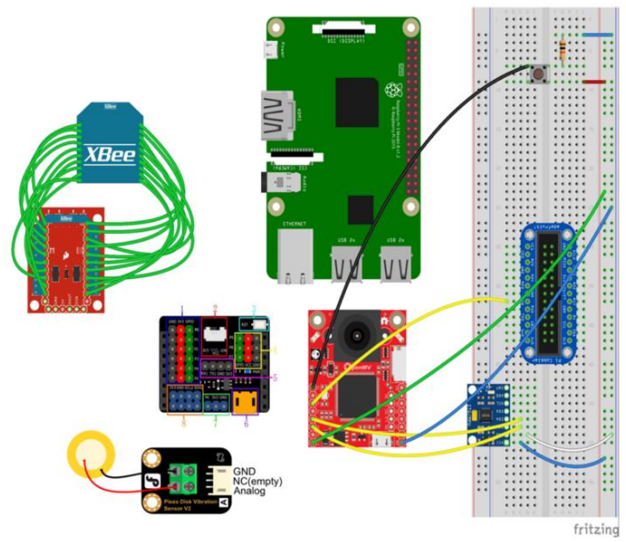
Photos

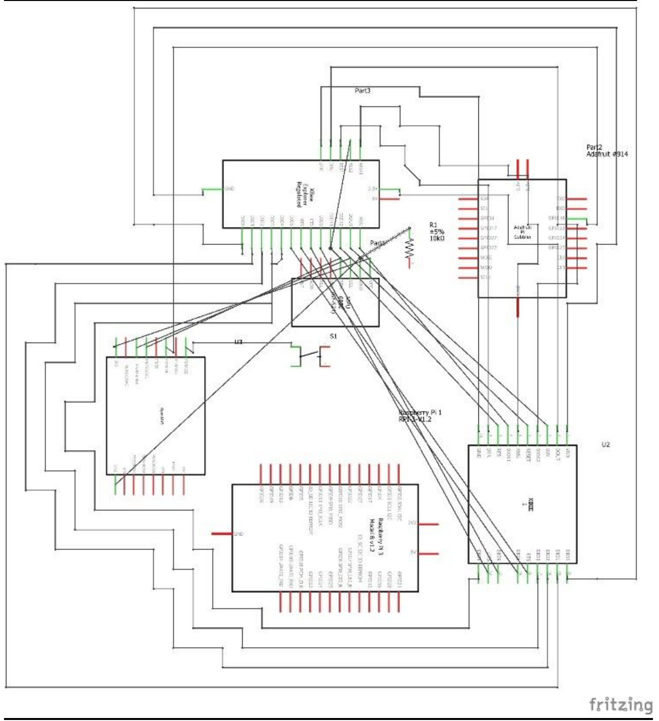





bern

Circuit/Block Diagram





The schematic doesn't include the IO Expansion Shield or the Piezo Disk Vibration Buzzer because these devices don't exist in Fritzing.

Code:

Code for openMV M7:

```
#Code primarily from openMV mjpeg example with some debouncing code from
#https://www.arduino.cc/en/Tutorial/Debounce

#Contains code from https://forums.openmv.io/viewtopic.php?t=718,
# https://www.electronicwings.com/raspberry-pi/mpu6050-accelerometergyroscope-
interfacing-with-raspberry-pi
# and https://docs.openmv.io/openmvcam/tutorial/analog_io.html that has been edited
to work with my
accelerometer

import sensor, image, time, mjpeg, pyb, math, utime, sys
from machine import I2C

RED_LED_PIN = 1
GREEN_LED_PIN = 2
BLUE_LED_PIN =  3

sensor.reset()                      # Reset and initialize the sensor.
sensor.set_pixformat(sensor.RGB565) # Set pixel format to RGB565 (or GRAYSCALE)
sensor.set_framesize(sensor.QVGA)   # Set frame size to QVGA (320x240)
sensor.skip_frames(time = 200)     # Wait for settings take effect.
clock = time.clock()                # Create a clock object to track the FPS.

pyb.LED(RED_LED_PIN).on()
sensor.skip_frames(time = 200) # Give the user time to get ready.

pyb.LED(RED_LED_PIN).off()

PWR_MGMT_1   = 0x6B
SMPLRT_DIV   = 0x19
CONFIG       = 0x1A
GYRO_CONFIG  = 0x1B
ACCEL_CONFIG = 0x1C
INT_ENABLE   = 0x38
ACCEL_XOUT_H = 0x3B
ACCEL_YOUT_H = 0x3D
ACCEL_ZOUT_H = 0x3F
GYRO_XOUT_H  = 0x43
GYRO_YOUT_H  = 0x45
GYRO_ZOUT_H  = 0x47
```

```
#for reading raspberry pi
rasp_pin = pyb.Pin('P2', pyb.Pin.PULL_UP)

issue = 1

#for accelerometer and gyroscope
accel_addr = 0x68

accelerometer = I2C(-1, "P5", "P4", freq=400000)
accelerometer.init("P5", "P4", freq=400000)

accel_buf = [0, 0, 0, 0, 0, 0, 0, 1]
acc_buf = bytearray(accel_buf)
accelerometer.writeto_mem(accel_addr, PWR_MGMT_1, acc_buf)


buf = [0, 0, 0, 1, 1, 0, 0, 0]
gen_buf = bytearray(buf)
accelerometer.writeto_mem(accel_addr, ACCEL_CONFIG, gen_buf)
accelerometer.writeto_mem(accel_addr, GYRO_CONFIG, gen_buf)

# helper for converting 2 bytes to int.
btoi = lambda msb, lsb: (msb << 8 | lsb) if not msb & 0x80 else -(((msb ^ 255) <<
8) | (lsb ^ 255) + 1)

accel_range = 16.0
accel_rate = 2048.0
gyro_range = 2000.0
gyro_rate = 16.4
accel = [0.0] * 3
gyro = [0.0] * 3
previous_gyro_x = None
previous_gyro_y = None
previous_gyro_z = None
previous_accel_z = None


# init buffers
adata_x = bytearray(6)
gdata_x = bytearray(6)
adata_y = bytearray(6)
gdata_y = bytearray(6)
```

```
adata_z = bytearray(6)
gdata_z = bytearray(6)

#for the vibration sensor
vibr = pyb.ADC(pyb.Pin('P6'))

#the video we are recording
movie = mjpeg.Mjpeg("recording.mjpeg")
button = pyb.Pin('P0', pyb.Pin.PULL_UP)
waittime = 500
previous_bounce_time = clock

last_state = 0;

checker = 0;

while(True):
    if (button.value()):
        value = 1
        start = pyb.millis()
        while(True):
            previous_bounce_time = 0;
            pyb.LED(BLUE_LED_PIN).on();
            clock.tick()
            movie.add_frame(sensor.snapshot())
            last_state = 1;
            #accelerometer and gyroscope code section
            accelerometer.readfrom_mem_into(accel_addr, ACCEL_XOUT_H, adata_x)
            accelerometer.readfrom_mem_into(accel_addr, GYRO_XOUT_H, gdata_x)
            accel_x = [btoi(adata_x[0], adata_x[1])/accel_rate, btoi(adata_x[2],
adata_x[3])/accel_rate, btoi(adata_x[4], adata_x[5])/accel_rate]
            gyro_x = [btoi(gdata_x[0], gdata_x[1])/gyro_rate, btoi(gdata_x[2],
gdata_x[3])/gyro_rate, btoi(gdata_x[4], gdata_x[5])/gyro_rate]
            accelerometer.readfrom_mem_into(accel_addr, ACCEL_YOUT_H, adata_y)
            accelerometer.readfrom_mem_into(accel_addr, GYRO_YOUT_H, gdata_y)
            accel_y = [btoi(adata_y[0], adata_y[1])/accel_rate, btoi(adata_y[2],
adata_y[3])/accel_rate, btoi(adata_y[4], adata_y[5])/accel_rate]
            gyro_y = [btoi(gdata_y[0], gdata_y[1])/gyro_rate, btoi(gdata_y[2],
gdata_y[3])/gyro_rate, btoi(gdata_y[4], gdata_y[5])/gyro_rate]
            accelerometer.readfrom_mem_into(accel_addr, ACCEL_YOUT_H, adata_z)
            accelerometer.readfrom_mem_into(accel_addr, GYRO_YOUT_H, gdata_z)
            accel_z = [btoi(adata_z[0], adata_z[1])/accel_rate, btoi(adata_z[2],
adata_z[3])/accel_rate, btoi(adata_z[4], adata_z[5])/accel_rate]
```

```
            gyro_z = [btoi(gdata_z[0], gdata_z[1])/gyro_rate, btoi(gdata_z[2],
gdata_z[3])/gyro_rate, btoi(gdata_z[4], gdata_z[5])/gyro_rate]

            #focusing on accel_z because it would note whether the rider is still
upright
            for i in range (0, 2):
                if (previous_accel_z != None):
                    if (previous_accel_z[i]  > accel_z[i] + 5):
                        sensor.snapshot().save("problem-%d.jpg" % issue)
                        issue+=1

                if (previous_gyro_x != None):
                    #focusing on all the gyros because it will tell us whether
there has been a slow down or increase in any axis
                    if (previous_gyro_x[i] > gyro_x[i] + 5):
                        sensor.snapshot().save("problem-%d.jpg" % issue)
                        issue+=1

                if (previous_gyro_y != None):
                    if (previous_gyro_y[i] > gyro_y[i] + 5):
                        sensor.snapshot().save("problem-%d.jpg" % issue)
                        issue+=1

                if (previous_gyro_z != None):
                    if (previous_gyro_z[i] > gyro_z[i] + 5):
                        sensor.snapshot().save("problem-%d.jpg" % issue)
                        issue+=1

            #pyb.delay(3000)
            previous_accel_x = accel_x
            previous_accel_y = accel_y
            previous_gyro_z = gyro_z

            #for the vibration sensor
            checker_value = ((vibr.read() * 3.3) + 2047.5) / 4095
            if(checker_value > 0.53):
                sensor.snapshot().save("problem-%d.jpg" % issue)
                issue+=1

            pyb.delay(50)

            #for dealing with alert notifications
            if(rasp_pin):
```

```
            sensor.snapshot().save("problem-%d.jpg" % issue)
            issue+=1
            pyb.LED(BLUE_LED_PIN).off()
            pyb.LED(GREEN_LED_PIN).on()
            pyb.delay(200)
            pyb.LED(GREEN_LED_PIN).off()
            pyb.LED(BLUE_LED_PIN).on()

        if(button.value() == last_state):
            if(start > waittime):
                    movie.close(clock.fps())
                    pyb.LED(BLUE_LED_PIN).off()
                    print("Done! Reset the camera to see the saved recording.")

                    pyb.hard_reset()
```

Code for the Raspberry Pi to read the Xbee and send a 1 to the camera:

```
# code from https://circuitdigest.com/microcontroller-projects/raspberry-pi-xbee-module-
interfacing

import time
import serial
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BOARD)
GPIO.setup(18, GPIO.OUT)

ser = serial.Serial(
port='/dev/ttyUSB0',
baudrate = 9600,
parity=serial.PARITY_NONE,
stopbits=serial.STOPBITS_ONE,
bytesize=serial.EIGHTBITS,
timeout=1
)

messages=ser.readline().strip()
if (messages == 1003):
GPIO.output(18, 1)

if (messages == 1004):
GPIO.output(18, 1)
```